FARAUTE, Vol. 6, No. 1, 11-19, 2011
ISNN 1698-7418
Deposito Legal PP200402CA1617

SkyHash: An evaluation algorithm for skyline queries over relational databases

Marlene Goncalves[1]*, Maria Esther Vidal [1]
[1] *Universidad Simón Bolívar. Departamento de Computación y Tecnología de la Información. Caracas, Venezuela*
***Autor de correspondencia**: mgoncalves@usb.ve

Resumen
Los lenguajes de consultas *Skyline* han sido definidos para permitir a los usuarios expresar sus criterios de preferencias. Una consulta *Skyline* recupera las tuplas no dominadas, es decir, un conjunto de tuplas tal que no existe otra tupla que sea mejor en todos los criterios de usuario. Distintas soluciones han sido propuestas para evaluar consultas *Skyline.* Esas soluciones pueden ser clasificadas en indexadas o de recorrido secuencial. Sin embargo, no tenemos conocimiento, de que exista alguna solución basada en *Hashing.* En este trabajo proponemos *SkyHash*, un algoritmo basado en *Hash* para consultas *Skyline. SkyHash* particiona los datos de entrada de acuerdo a una función *hash* y distribuye las tuplas no dominadas y algunas de sus dominadas en una misma partición dependiendo de los valores de la función *hash.* Así, una fase de filtrado puede ser ejecutada para descartar tuplas dominadas rápidamente. Empíricamente, estudiamos el comportamiento de *SkyHash.* Nuestros resultados iniciales muestran que el tiempo de *SkyHash* puede ser considerablemente menor que el tiempo de ejecución de los algoritmos del estado del arte.

Palabras Claves: Consultas Basadas en Preferencias, Skyline, SQL

Abstract
Skyline query languages have been defined to allow users to express their preference criteria. A Skyline query retrieves non-dominated tuples, i.e., a set of tuples such that does not exist another one that is better for all user criteria. Several solutions have been proposed in order to evaluate Skyline queries. These solutions can be classified as indexed or scan-based. To the best of our knowledge, none of the existing solutions are based on Hashing. In this work, we propose SkyHash, a hash-based algorithm for Skyline queries. SkyHash partitions the input data according to a hash function; it distributes non-dominated tuples and some of their dominated ones in the same partition depending on the hash function values. Thus, a filtering phase can be performed to discard dominated tuples quickly. Empirically, we study the behavior of SkyHash. Our initial results show that the execution time of SkyHash may be considerably less than the execution time of state-of-art algorithms.

Keywords: Preference Based Queries, Skyline, SQL

1. Introduction

SQL (Structured Query Language) is the standard querying language for relational databases. However, SQL has not a special clause for specification of user preferences. If a user wants to express his preferences through SQL, he must determine the order in which the answer will be produced by using the ORDER BY clause and then, choose the ones that satisfy his preferences. This process can be tedious for those queries characterized by many preference criteria and results.

Thus, (Börzönyi *et al.*, 2001) introduced a SQL extension, known as Skyline queries, which easily allow the specification of user preferences. A Skyline query identifies a set of non-dominated tuples. A tuple *a* is said to dominate another tuple *b*, if a is as good or better than b for all attributes and strictly better than b in, at least, one.

To identify the Skyline, the existing evaluation algorithms have to compare each tuple against other tuples in order to probe that it is non-dominated (dominance probes). These evaluating algorithms can be classified as indexed or scan-based.

On one hand, indexed Skyline evaluation algorithms either use indexes on a set of dimensions or index pre-computed Skyline for efficient access to data. This type of algorithms is not regarded in this work. First, computational effort in Skyline evaluation is on dominance probes mainly. Second, if index structures are defined on base tables, the Skyline may not be computed using indexes from a set of materialized tuples. Third, although the Skyline can be pre-computed through indexes, the Skyline of a set of attributes may not be identified from the skylines of subsets of attributes, and vice versa. Finally, the index will be recalculate when a new tuple is inserted into the database such that it dominates some pre-computed Skyline tuples (Chomicki *et al.*, 2003; Godfrey *et al.*, 2003).

On the other hand, scan-based Skyline evaluation algorithms do not depend on index structures. They scan the entire table and select the non-dominated tuples. They may sort data (Chomicki *et al.*, 2003; Godfrey *et al*, 2003) for reducing the number of dominance probes. Moreover, they may discard dominated tuples while they read each page into memory (Godfrey *et al.*, 2003). Nevertheless, non-dominated and their dominated tuples can be in distinct pages so that none of tuples is discarded while sorting the input data.

To the best of our knowledge, none of the existing Skyline evaluation algorithms uses Hashing. In this work, SkyHash is proposed as a new algorithm based on Hashing in order to compute the Skyline that answers a query. By means of a hash function evaluation, each tuple is stored in one of several partitions so that non-dominated and some of their dominated tuples fall in the same bucket. Thus, SkyHash may discard some tuples in the same bucket initially before of sorting the input data. Additionally, an experimental study on SkyHash is presented in this work.

This paper comprises five sections. In Section 2, we introduce background and related works for Skyline queries. In Section 3, we describe SkyHash algorithm and illustrate how it works using an example for a two-dimensional Skyline query. In Section 4, we present an initial experimental study that shows the properties of the implemented SkyHash algorithm. Finally, in Section 6, we point out concluding remarks and future works.

2. Related Work and Background

Skyline query languages were proposed as a SQL extension in (Börzönyi *et al.*, 2001) that allows to specify user preferences through a multi-criteria function defined into the `SKYLINE OF` clause. Syntactically, a Skyline query may be expressed as:

```
SELECT < attributes >
  FROM < relations >
 [WHERE < conditions >]
[GROUP BY < attributes >]
 [HAVING < conditions >]
[SKYLINE OF a1 [MIN | MAX | DIFF],...,an
          [MIN | MAX | DIFF];
```

Where, the dimensions $a1, ..., an$ are the attributes that user preferences range over and their domains are integers, floats or dates. The `MIN` and `MAX` directives specify whether the user prefers lowest or highest values, respectively; the `DIFF` directive defines the interest in retaining the best choices with respect to every distinct value of that attribute.

The result of a Skyline query execution will be composed of all non-dominated tuples. A tuple is non-dominated if does not exist another tuple equally

good or better in all dimensions and better in at least one.

Several efforts have been made to develop Skyline evaluation algorithms in a relational database context. Existing algorithms can be classified as indexed or scan-based algorithms (Tan *et al.*, 2001; Kossman *et al.*, 2002; Papadias et al., 2005; Chen & Lian, 2008; Fuhry et al., 2009). Tan et al.'s algorithm, NN (Nearest Neighbor) and BBS (Branch-and-Bound Skyline) are indexed evaluation algorithms that progressively return each Skyline tuple without necessarily scanning all the tuples. However, this work does not regard indexed evaluation algorithms due primarily to the dominance probes -instead efficient access to data- affect Skyline performance strongly. On the other hand, Block-Nested-Loops (BNL) (Börzönyi *et al.*, 2001), Sort-Filter-Skyline (SFS) (Chomicki *et al.*, 2003) and LESS (Linear Elimination Sort for Skyline) (Godfrey *et al*, 2003) are three scan-based evaluation algorithms that allow to identify the Skyline in relational database systems. The BNL algorithm scans the entire table while it maintains a window of non-dominated tuples, which could be replaced by any other tuple that is seen later on. A BNL variant is SFS which only requires a previous sorting step in terms of a topological order compatible with the Skyline criteria and it does not need window tuple replacement like BNL, because of the mentioned initial topological sort. SFS algorithm begins sorting the table, after it passes a cursor over the sorted rows and it finally discards dominated rows. SFS improves the effectiveness of dominated discarding by means of an entropy function (Chomicki *et al.*, 2003) over a tuple t according to the Eq. (1).

$$E(t) = \sum_{i=1}^{K} \ln\left(t[a_i] + 1\right) \quad (1)$$

where $t[a_i]'$s are the normalized Skyline dimensions and k is the number of dimensions. A tuple whose entropy function value is high, probably discards more tuples per pass because it may dominate many tuples (Chomicki *et al.*, 2003). Thus, data is sorted by this entropy function for discarding dominated tuples quickly.

The algorithm LESS initially sorts tuples as SFS does, but presents two improvements over it: in the first ordering phase, it uses an elimination-filter window to discard dominated tuples quickly and it combines the last phase of the sort algorithm with the Skyline filter phase of SFS to eliminate remaining dominated tuples.

RAND is the first randomized streaming algorithm for skyline query evaluation (Das Sarma *et al.*, 2009). It was thought to compute the skyline of a massive database with a good worst case guarantee.

The problem of group-by skyline queries has also been addressed. A group-by skyline query is a query that contains a DIFF directive in the SKYLINE OF clause. Few works focus on group-by skyline queries. (Luk *et al.*, 2009) present a study on algorithms to be implemented in commercial database systems in order to evaluate group-by Skyline queries.

Our work focuses on relational context. Nevertheless, Skyline has also been widely studied in other contexts and applications. In distributed Web systems, data are available through Web sites. Efficient algorithms to compute distributed Skyline queries have been developed. These algorithms consider sequential and random accesses. BDS (Basic Distributed Skyline) is one of these algorithms and was defined by (Balke *et al.*, 2004). BDS retrieves a Skyline superset in a first phase and it discards the dominated objects from this superset in a second phase. (Balke & Güntzer, 2004) introduced another algorithm to calculate Skyline distributed queries. This algorithm constantly updates a virtual object which is compared against each accessed object until an accessed object dominates this virtual object. PDS (Progressive Distributed Skyli-ning) is an algorithm for Progressive Skyline queries (Lo *et al.*, 2006). PDS is based on a multidimensional index to allow retrieving progressively Skyline objects on the Web.

(Huang *et al.*, 2006) also presented algorithms to evaluate Skyline queries but in mobile distributed environments using devices that store horizontally partitioned data in ad hoc networks. The queries are characterized by geographical search regions of interest and are evaluated on limited capacity devices.

Additionally, new solutions have been created for situations where the user selects

a subset or subspace of the attributes according to his interests or

the user wants to know all the subspaces to better understand data semantics (Pei *et al.*, 2005; Yuan *et al.*, 2005; Tao *et al.*, 2006).

(Pei *et al.*, 2005; Yuan *et al.*, 2005) independently proposed SKY-CUBE, which is a lattice structure composed of Skylines in all subspaces. SUBSKY is

a technique to calculate the Skyline of a particular subspace (Tao *et al.*, 2006). SUBSKY uses a transformation that converts each user preference to one-dimensional value that is indexed by a B-tree. (Pei *et al.*, 2007) introduced a new efficient algorithm for Skyline cube computation without enumerating all subspaces. Finally, (Raissi *et al.*, 2010) defined a novel algorithm to efficiently calculate the SKYCUBE. This algorithm reduces dominance probes in one subspace and the number of subspaces to be searched.

Lastly, variants of Skyline have been proposed to several multi-criteria decision making applications. (Wong *et al.*, 2008) based their work on evaluation of Skyline queries whose preferences are dynamic functions and attributes are categorical or nominal; categorical data consist of few values corresponding to a category value o label. A reverse Skyline query (Wu *et al.*, 2009) is a variant of Skyline which retrieves all objects whose Skyline contains a query object. Suppose a set U of used cars, a set C of customers, and a used car X. A reverse skyline returns all customers in C that do not find any other used car in U to be better than X.

3. Motivating Example

Consider a restaurant guide where each restaurant is described by an identifier (*idRest*), a name (*Name*), the average dish price (*AvegPrice*), and two values between 1 and 80 that measure the quality of the service (*QService*) and the quality of food (*QFood*), respectively.

Suppose that the best or Skyline restaurants are defined in terms of the values of *QService*and *QFood.* A Skyline restaurant will be selected if and only if there is no other restaurant with better quality of service and food. To choose a Skyline restaurant, one must identify the set of all the restaurants that are non-dominated by any other restaurant in terms of these criteria.

Fig. 1 shows a graphical representation for points corresponding to restaurant tuples. The bold points represent those Skyline restaurants which are supposed to have highest quality of service and food. The other restaurants are not Skyline because they are dominated, i.e., there is at least one of the Skyline restaurants that has better values on the attributes *QService* and *QFood.*

We can partitionate the data space into disjointed regions. Suppose the origin point $O = (0,0)$ and four regions such as Fig. 2 depicts. We can imagine three vectors v_1, v_2 and v_3 whose origin point is O and then, find their angles with respect to a fixed vector $f = (80,0)$. Thus, data space is divided into the following four regions: • Region 1 contains

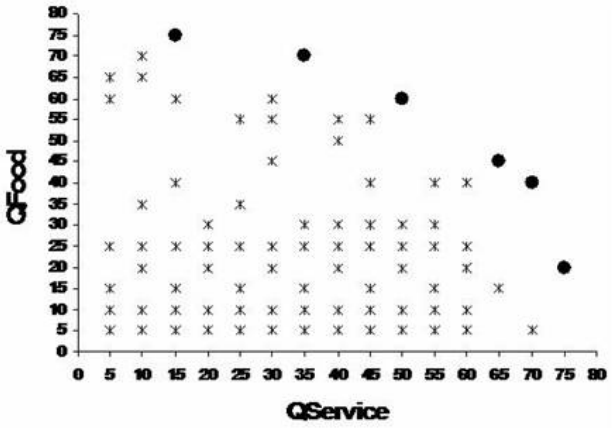


Figura 1. Graphical representation of restaurants

all vectors that are between f and v_1, i.e., those that have an angle between 0 and $\pi/8$.

- Region 2 contains all vectors that are between v_1 and v_2, i.e., those that have an angle between $\pi/8$. and $\pi/4$.
- Region 3 contains all vectors that are between v_2 and v_3, i.e., those that have an angle between $\pi/4$. and $3\pi/8$.
- Region 4 contains all vectors that have an angle between $3\pi/8$. and $\pi/2$.

Each region in Fig. 2 includes Skyline points and some of their dominated ones. Once data space is divided, the first dominated points may be discarded and for each region, partial skylines may be identified. Clearly, these dominated points will not be compared against others because they have already been filtered. Moreover, fewer points will have to be probed in order to identify the final Skyline in the next step.

We have now intuitively evaluated a Skyline query. On one side, partial skylines were calculated by regions in order to discard dominated tuples quickly. On the other side, fewer tuples will be probed to return the Skyline that answers the query. If a disk page does not contain dominated tuples, algorithms, such as LESS, may not filter the dominated ones in its first ordering phase. With SkyHash, the in-

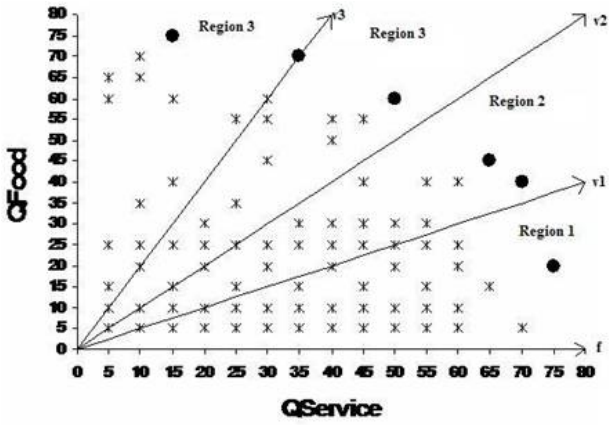


Figura 2. Regions of data space

put data is divided by regions so that non-dominated and some of their dominated tuples fall in the same region. Therefore, SkyHash may filter some tuples in the same region initially.

4. The SkyHash Algorithm

Before presenting SkyHash, we briefly describe the differences between BNL, SFS, LESS and SkyHash. The algorithm BNL may replace tuples in the window while scans a table. To avoid tuple replacement, SFS sorts tuples in terms of an entropy function. Basically, the algorithm SFS consists of two steps. The first step sorts data by an entropy function and the second step scans sorted data discarding dominated tuples. Finally, LESS improves SFS because eliminates dominated tuples while it is sorting data and therefore, fewer tuples are compared pair-wise at the end.

Similarly to LESS, SkyHash modifies an External Merge Sort algorithm in order to sort tuples according to the entropy function. During execution, the algorithms merge the runs and write to the output buffer page while there are runs to be merged from the previous pass. The first and the last phase of merge are adapted to filter dominated tuples.

In the first ordering phase, LESS reads each page into memory, sorts it by the entropy function and discards dominated tuples for each page. Here, it is possible that dominated tuples are not in the same page as their dominators due to data storage. In this case, none of dominated tuples will be discarded.

Unlike LESS, SkyHash evaluates a hash function h to select a suitable partition for each tuple. The tuples will be distributed among several partitions. Each partition will contain non-dominated tuples and some of their dominated ones. Once data are partitioned, SkyHash discards the dominated tuples by partition and then, merge the partitions and eliminate remaining dominated tuples.

The SkyHash is presented in Algorithm 1. In step 2 (**INITIALIZE**), n partitions are initialized and the number of partitions is computed. In step 3 (**PARTITION**), it scans the table T and evaluates a hash function for each tuple. Each tuple is inserted in a partition according to the hash function. The hash function calculates angle Θ for the vector that corresponds to tuple t and determines its partition number. Step 9 (**MERGE**) eliminates dominated tuples in each partition, merges them, and finally discards remaining dominated tuples. All partitions are sorted by the entropy function.

For simplicity, we suppose that dimensions related to multi-criteria function are minimized. If some dimension a_i is maximized, the angle is computed by subtracting a_i from the highest value of a_i, this is, the maximum value of a_i minus a_i.

1. `Skyline SkyHash(Table T,`
   `Function m,integer n) {`
2. $P_1, ..., P_n \leftarrow \varnothing$;
3. $r \leftarrow \Theta/2n$;
4. `While (exists p belongs`
   `to the set of pages P`
5. `Select the next page p` ε `P;`
6. `For each tuple t of p`
7. `Calculate its angle` Θ;
8. $j \leftarrow \lceil(\Theta - r)/r\rceil$;"
9. `Add t to the partition Pj;`
10. `For each partition`
11. `find those tuples dominated`
    `by some other tuples in the`
    `same partition;`
12. `Utilize an External Merge`
    `Sort algorithm;`
13. `Eliminate remaining dominated`
    `tuples in the last`
    `ordering phase;`
14. `return Skyline tuples;`
15. }

Algorithm 1 SkyHash Algorithmm

5. Initial Experimental Study

The experimental study consisted of experiments running over relational tables of 10,000 tuples. Each table contained an identifier and six columns whose values varied from 0.0 to 1.0. Any column might have duplicated values. The attribute values were generated for the following data distribution:

- Mixed: Attributes are independent of each other. Data were divided into two groups of three columns: one group was generated using a uniform distribution and the other, using a Gaussian distribution from five overlapping multidimensional Gaussian Bells.
- Correlated: Attributes are independent of each other. Data were divided in two correlated groups of three real columns. In each group, the attribute values for a base column were generated following a uniform distribution.
- Uniform: Attributes are independent of each other and their values are generated uniformly.

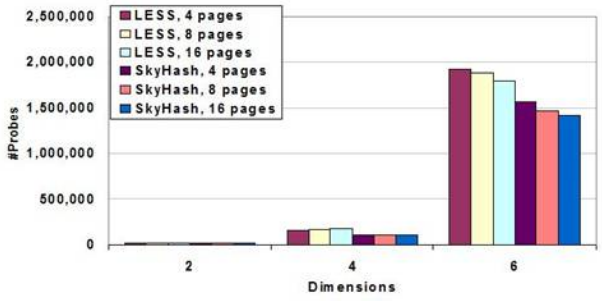
We randomly generated ten queries characterized by the following properties:

1. Only one table in the FROM clause;
2. The attributes in the multi-criteria function were chosen randomly among the attributes of the table, following a uniform distribution;
3. Directives for each attribute of the multi-criteria function were selected randomly considering only maximizing and minimizing criteria;
4. The number of attributes of the multi-criteria function was two, four and six.

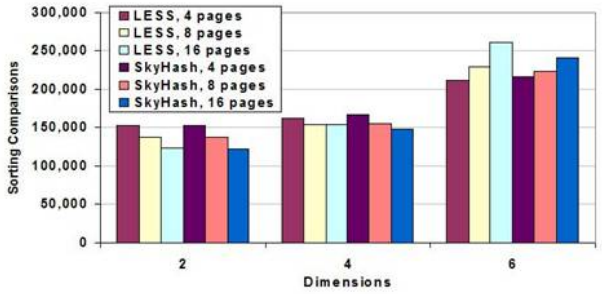
The algorithms LESS and SkyHash were implemented in Java 1.5, the data were stored into Oracle 9i, and the experiments were executed on a SunFire V440 machine equipped with 2 processors Sparcv9 of 1.281 MHZ, 16 GB of memory and 4 disks Ultra320 SCSI of 73 GB running on SunOS 5.10 (Solaris 10). Finally, the number of pages for sorting was 4, 8 and 16.

BNL and SFS are not considered because LESS outperforms them. However, an experimental study on BNL, SFS and LESS is presented in (Chomicki *et al.*, 2003; Godfrey *et. al*, 2003). RAND is neither regarded because it is comparable than LESS in the average case (Raïssi *et al.*, 2010).

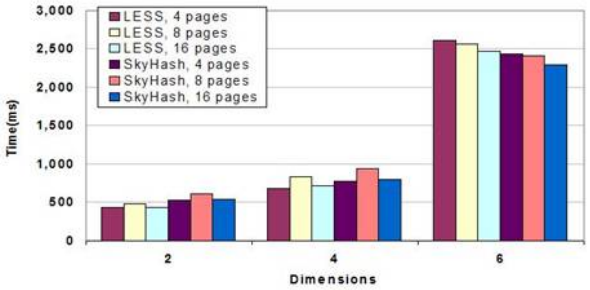
Fig. 3 reports the average of dominance probes, the average of comparisons by sorting and average time in milliseconds, respectively, used by each algorithm for the mixed dataset.



(a) Average of Dominance Probes



(b) Average of Sorting Comparisons (ms)



(c) Average Time (ms)

Figura 3. Results for Mixed Dataset

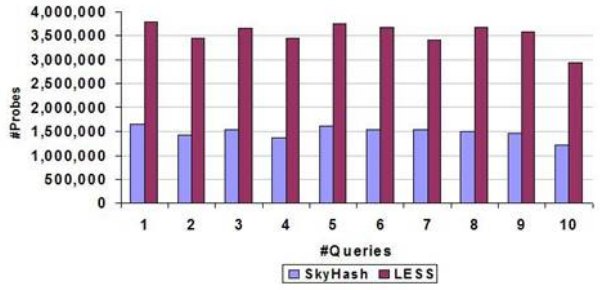
In general, we observed that:

1. The average of dominance probes is lower for SkyHash. The reason of this behavior may be because this algorithm inserts non-dominated tuples and some of their dominated ones in the same partition more adequately than LESS. Thus, more tuples were filtered initially, and fewer tuples were compared.
2. The number of dominance probes decrease as the number of pages increases. If there are more pages, more dominated tuples will be eliminated because it is possible that there exist more partial skylines that dominate them.

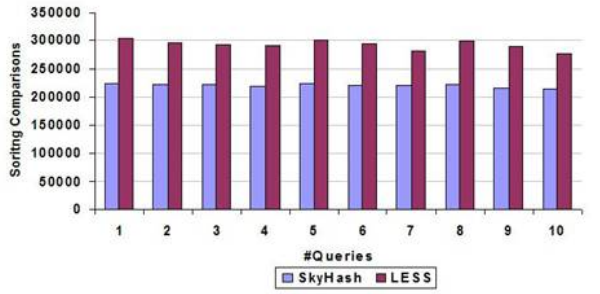
3. The average of comparisons by sorting is lower for SkyHash. This behavior indicates that more tuples were discarded initially and therefore fewer tuples are being merged and sorted.

4. The LESS algorithm requires more time. This behavior is because SkyHash makes fewer dominance probes and sorting comparisons than LESS.

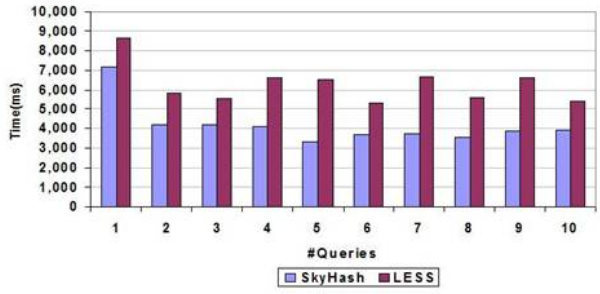
We also ran experiments for ten 4-dimensional queries on a uniform dataset where each page does not contain dominated tuples. Fig. 4 reports for each query the number of dominance probes, the number of comparisons by sorting and time in milliseconds.



(a) Average of Dominance Probes



(b) Average of Sorting Comparisons



(c) Average Time (ms)

Figura 4. Results for Generated Dataset

In general, we can observe that SkyHash outperforms LESS. In this case, angle calculation is a hash function that contributes to distribute better the tuples than LESS. Thus, less number of dominance probes and comparisons by sorting were performed by SkyHash.

Finally, in Table 1 we report the average of: the number of dominance probes and time in milliseconds for 10 queries on uniform data.

Table 1 Results for the algorithms LESS and SkyHash

	Dominance Probes		Time (ms)	
	LESS	*SkyHash*	*LESS*	*SkyHash*
Average	685746	538176	1312	1186
t-test	p-value=0.0191426		p-value=0.0000011	

In Table 1, we may observe that SkyHash evaluates fewer dominance probes and sorting comparisons than LESS. Since the initial redistribution where each partition contains non-dominated tuples and some of their dominated ones, some tuples are filtered and fewer tuples are compared and sorted in the last step. In consequence, Skyhash time is less than LESS.

Table 1 also presents the results for the t-test. As this analysis shows, the differences for the number of probes, sorting comparisons and required time are highly significant (at least 99.99 % level). A high significance level indicates there is almost certainly a true difference in required time and the number of dominance probes performed by both algorithms.

6. Conclusions and Future Work

In this work, we have proposed SkyHash as a new algorithm that computes the Skyline using a hash function. Although the LESS behavior is similar to SkyHash, LESS cannot discard tuples during the first pass when each page contains incomparable tuples and their dominated one in the same page.

Our experimental results show the number of probes performed by dominance and sorting is reduced because, more tuples are filtered before merging phase. The data re-distribution based on a hash function implies that dominated tuples are quickly discarded. Consequently, there is a computational saving in processing time when the Skyline is computed using SkyHash.

In the future, we plan to include SkyHash into a relational engine for integration with other basic relational operators. Additionally, we plan to experimentally evaluate the performance of RAND and

SkyHash. Our experimental study does not involve the RAND algorithm because it is comparable to LESS in the average case. However, RAND has well behavior in the worst case.

Lastly, few works have been addressed the problem of evaluation of group-queries Skyline. We also want to experimentally study this kind of queries using our algorithm.

7. References

Balke, W. & U. Güntzer. (2004). Multi-objetive query processing for database systems. Proceedings of the International Conference on Very Large Databases (VLDB). Toronto. Canada. 936.

Balke, W., U. Güntzer & J. Zheng. (2004). Efficient distributed skylining for web information systems. Proceedings of the International Conference on Extending Database Technology (EDBT). 256.

Börzönyi, S., D. Kossman & K. Stocker. (2001). The Skyline operator. Proceedings of the International Conference on Data Engineering (ICDE). IEEE Computer Society. Washington. USA. 421.

Chen, L., & X. Lian. (2008). Dynamic skyline queries in metric spaces. Proceedings of the International Conference on Extending Database Technology (EDBT). New York. USA. 333.

Chen, L., & X. Lian. (2008). Dynamic skyline queries in metric spaces. Proceedings of the International Conference on Extending Database Technology (EDBT). New York. USA. 333.

Chomicki, J., P. Godfrey, J. Gryz & D. Liang. (2003). Skyline with presorting. Proceedings of the International Conference on Data Engineering (ICDE). 717.

Das Sarma, A., A. Lall, D. Nanongkai & J. Xu. (2009). Randomized Multi-pass Streaming Skyline Algorithms. Proceedings of the Vldb Endowment (PVLDB). 2 (1): 85-96.

Fuhry, D., R. Jin & D. Zhang. (2009). Efficient skyline computation in metric space. . Proceedings of the International Conference on Extending Database Technology (EDBT). New York. USA. 1042.

Godfrey, P., R. Shipley & J. Gryz. (2005). Maximal Vector Computation in Large Data Sets. Proceedings of the Conference on Very Large Data Bases (VLDB). trondheim, Norway. 229.

Huang, Z., C. Jensen, H. Lu & B. Ooi. (2006). Skyline queries against mobile lightweight devices in manets. Proceedings of International Conference on Data Engineering (ICDE). IEEE Computer Society. Washington. USA. 66.

Kossman, D., F. Ransak & S. Rost. (2002) Shooting stars in the sky: An online algorithm for Skyline queries. Proceedings of the International Conference on Very Large Databases (VLDB). 275.

Kung, H., F. Luccio, F. Preparata. (1975). On finding the maxima of a set of vectors. J. ACM. **22**(4): 469-476.

Lo, E., K. Yip, K. Lin & D. Cheung. (2006). Progressive skylining over web-accessible database. J. Data Knowl. Eng. **57**(2): 122-147.

Luk, M., M. Yiu & E. Lo. (2009). Group-by skyline query processing in relational engines. Proceedings of the 18^{th} ACM Conference on Information and Knowledge Management (CIKM). Hong Kong. China. 1433.

Papadias, D., Y. Tao, G. Fu & B. Seeger. (2005). Progressive Skyline computation in database systems. ACM Trans. Database Syst. **30**(1):41-82.

Pei, J., W. Jin, M. Ester & Y. Tao. (2005). Catching the best views of skyline: A semantic approach based on decisive subspaces. Proceedings of the 31st International Conference on Very Large Data Bases (VLDB). Trondheim. Norway. 253.

Pei, J., A. Fu, X. Lin & H. Wang. (2007). Computing compressed multidimensional skyline cubes efficiently. Proceedings of the International Conference on Data Engineering (ICDE). 96.

Raïssi, R., J. Pei & T. Kister. (2010). Computing Closed Skycubes. Proceedings of the 36^{th} International Conference on Very Large Data Bases (VLDB). Singapure. 838.

Tan, K., P. Eng & B. Ooi. (2001). Efficient progressive Skyline computation. Proceedings of Conference on Very Large Data Bases. 301-310.

Tao, Y., Xiao, X. and Pei, J. (2006). Subsky: Efficient computation of skylines in subspaces. Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE), 65-74.

Wong, R. Ch-W., Fu, A. W-Ch., Pei, J., Ho, Y. S., Wong, T. and Liu, Y. (2008). Efficient skyline

querying with variable user preferences on nominal attributes. Proceedings of the VLDB Endowment (PVLDB) **1**(1), 1032-1043.

Wu, X., Tao, Y., Wong, R. C., Ding, L., Yu, J. X. (2009). Finding the Influence Set through Skylines. Proceedings of 12^{th} International Conference on Extending Database Technology (EDBT), 1030-1041.

Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J. X. and Zhang, Q. (2005). Efficient computation of the skyline cube. Proceedings of the 31st international conference on Very large data bases (VLDB), 241-252.